Spot the Error: Non-autoregressive Graphic Layout Generation with Wireframe Locator

Jieru Lin[1]*, Danqing Huang[2], Tiejun Zhao[1], Dechen Zhan[1], Chin-Yew Lin[2]
[1]Harbin Institute of Technology, Harbin, China
[2]Microsoft Research
hitjierulin@gmail.com, {dahua, cyl}@microsoft.com, {tjzhao, dechen}@hit.edu.cn

Abstract

Layout generation is a critical step in graphic design to achieve meaningful compositions of elements. Most previous works view it as a sequence generation problem by concatenating element attribute tokens (i.e., category, size, position). So far the autoregressive approach (AR) has achieved promising results, but is still limited in global context modeling and suffers from error propagation since it can only attend to the previously generated tokens. Recent non-autoregressive attempts (NAR) have shown competitive results, which provides a wider context range and the flexibility to refine with iterative decoding. However, current works only use simple heuristics to recognize erroneous tokens for refinement which is inaccurate. This paper first conducts an in-depth analysis to better understand the difference between the AR and NAR framework. Furthermore, based on our observation that pixel space is more sensitive in capturing spatial patterns of graphic layouts (e.g., overlap, alignment), we propose a learning-based locator to detect erroneous tokens which takes the wireframe image rendered from the generated layout sequence as input. We show that it serves as a complementary modality to the element sequence in object space and contributes greatly to the overall performance. Experiments on two public datasets show that our approach outperforms both AR and NAR baselines. Extensive studies further prove the effectiveness of different modules with interesting findings. Our code will be available at https://github.com/ffffatgoose/SpotError.

Introduction

Layout generation refers to the arrangement of elements (i.e., size and position) on a canvas, which is essential for creating visually appealing graphic designs (e.g., articles, user interface). State-of-the-art systems (Jyothi et al. 2019; Arroyo, Postels, and Tombari 2021; Kikuchi et al. 2021; Yang et al. 2021) mostly view the task as a sequence generation problem where the sequence is composed of element attribute tokens (i.e., category, position, size). Majority of the works follow the autoregressive (AR) approach which generates one token at a time based on the previously output and have achieved promising results (Yang et al. 2021; Guo, Huang, and Xie 2021; Yang et al. 2023; Weng et al. 2023).

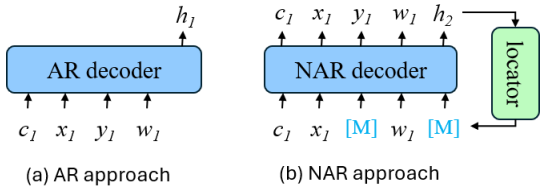

Figure 1: Illustration of (a) AR and (b) iterative-based NAR approach in layout generation. The AR approach generates one token at a time conditioned on previously generated tokens. NAR generates all tokens simultaneously and uses a locator (usually heuristics) to detect erroneous tokens which will be masked and re-predict in the next decoding iteration.

However, for layout generation, it is important to model relationships between elements as well as the global context. The inherent causal attention in the AR approach exposes limitation in global context modeling and causes immutable dependency chain issue and error propagation (Kong et al. 2022).

Recently, there are some attempts of non-autoregressive (NAR) approaches (Kong et al. 2022; Zhang et al. 2023; Inoue et al. 2023; Hui et al. 2023) that generate the entire sequence simultaneously. It allows a more flexible modeling of bidirectional context and shows promising results in the task. In this paper, we conduct an in-depth analysis to compare the AR and NAR approaches (as shown in Figure 1) with two findings: (1) we show that NAR is more robust to different element orders in the sequence than AR; (2) similar to the word repetition issue in non-autoregressive machine translation (NMT) (Huang, Perez, and Volkovs 2022), NAR tends to position elements to similar regions that will cause overlap problem and results in inferior layouts.

Iterative decoding is an effective mechanism to alleviate this issue. In the recent state-of-the-art NAR system BLT (Kong et al. 2022), it uses a Transformer-based decoder (Vaswani et al. 2017) and applies a heuristic strategy to choose the least confident tokens from the decoder's prediction as the erroneous ones. However, such simple heuristic is likely to be biased to the decoder's learned distribution and potentially propagate the mistakes, acting as noises to make a negative impact on subsequent iterations.

This paper proposes a learning-based locator module to

*Work was done during the first author's internship at Microsoft mentored by Danqing Huang.

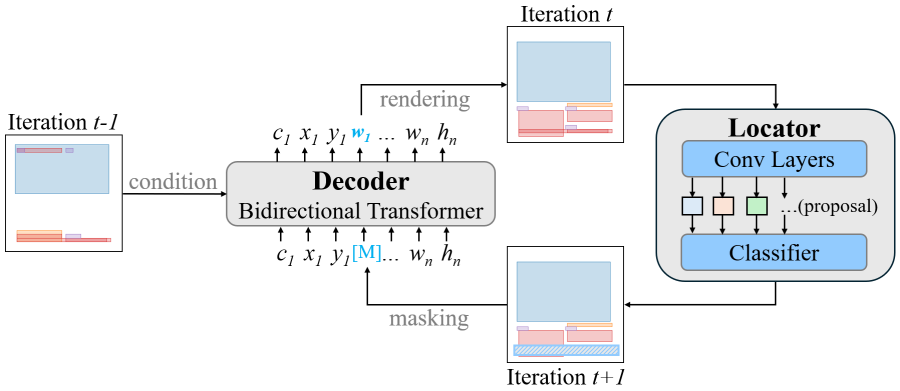

Figure 2: Overview of our pipeline. Our model consists of a decoder and a locator. For each mask-predict iteration, the locator detects the erroneous attribute tokens to be masked and the decoder predict the masked tokens in a non-autoregressive approach.

explicitly distinguish the incorrect tokens. Using a toy experiment of comparing different layout representations, we observe that pixel space is more sensitive in capturing spatial patterns of layouts (e.g., overlap, alignment). Therefore, our locator receives the input of wireframe image rendered from the decoder's generated sequence in the previous iteration, and then detects the erroneous element and classify the corresponding attribute tokens if they should be masked or not. The wireframe serves as a complementary modality to the layout sequence in object space which we will show is effective. To train the locator, it requires the annotation of erroneous attributes in a noisy layout (e.g., the width of an element is inaccurate and should be masked). One possible solution is that we make random noise on the real layouts without human labeling. However, our initial experiment shows minor improvement when applying the locator to the refinement process. In order to focus more on the decoder's error types and align with its distribution, we propose a novel automatic data construction pipeline. Specifically, we use the decoder's generated layout to retrieve the most similar real layout as ground truth and apply Hungarian matching between the two sets of elements. For the matched element pairs that with significant attribute value differences, we annotate the corresponding tokens as the targets to be masked.

We evaluate our approach on two public datasets related to graphic design: RICO (Deka et al. 2017) for mobile app UIs and PubLayNet (Zhong, Tang, and Yepes 2019) for scientific articles. Experiment results show that our approach outperforms current AR and NAR baselines in terms of both quantitative and qualitative evaluations. Furthermore, we conduct extensive analysis to better understand what our model has learned.

To summarize, the contributions of this paper include:

- We conduct an in-depth analysis to compare between AR and NAR models in layout generation. Though NAR is robust to different input element orders, it exposes the issue of repetitive token generation.
- We investigate the use of different layout representations (i.e., object, pixel) with a toy experiment. We observe that pixel space is more sensitive in capturing spatial patterns than object space, which motivates us to incorporate the image modality into the modeling process.
- To improve the iteration-based NAR, this paper proposes a learning-based locator that takes a rendering wireframe as input and detects erroneous tokens more accurately.

Related Works

Layout Generation

Layout generation is an important task in graphic design intelligence (Huang et al. 2023; Li et al. 2021) (e.g., layout representation learning (Xie et al. 2021; Feng et al. 2022), layout reverse engineering (Hao et al. 2023; Shi et al. 2023; Zhu et al. 2024; Huang et al. 2021)). Traditional works (Hurst, Li, and Marriott 2009; Kumar et al. 2011; O'Donovan, Agarwala, and Hertzmann 2014; Tabata et al. 2019) are mostly based on heuristics with constraint optimization, which usually ensure high-quality but limited outputs. Recently there are an increase in works based on deep neural networks. LayoutGAN (Li et al. 2019) applies GAN to synthesize the layout bounding box and proposes a differential wireframe rendering module to enable the training of discriminator, and LayoutGAN++ (Kikuchi et al. 2021) extends LayoutGAN with Transformer backbone. LayoutVAE (Jyothi et al. 2019) trains two VAEs separately, one to predict the element categories and the other to generate layouts given the category condition. Several methods also follow the VAE-based generative framework (Lee et al. 2020; Jiang et al. 2022). Recent works (Yang et al. 2021; Arroyo, Postels, and Tombari 2021) build generative backbone based on Transformer (Vaswani et al. 2017) to model long-distance dependency and perform better.

BLT (Kong et al. 2022) is the first recent attempt of using non-autoregressive in layout generation. The decoder is trained using BERT-like strategy to predict some randomly masked tokens. During inference, it uses iterative decoding that tokens with the least confident score are masked and further refinement. Diffusion-based models also follow the NAR framework (Inoue et al. 2023; Zhang et al. 2023; Hui et al. 2023) which progressively infers a complete layout from a noisy status with discrete diffusion process. Despite

different modeling approaches, the above mentioned methods use element sequence to represent a layout. This paper observes that layouts represented in pixel space offers a different aspect of features and hence we propose to combine the object space and pixel space in the NAR framework for better performance.

There are also some content-aware generation methods (Zheng et al. 2019; Wang et al. 2022; Cao et al. 2022; Zhou et al. 2022; Li, Zhang, and Wang 2022; Vaddamanu et al. 2022) that further considers the element content into modeling, which we will leave for future exploration.

Non-autoregressive Sequence Generation

Comared to AR (Huang et al. 2018a; Zhou and Huang 2019; Huang et al. 2018b), NAR is more efficient in inference and has been applied in various tasks such as neural machine translation, automatic speech recognition and text to speech. Here we mainly focus on the related works of iteration-based non-autoregressive machine translation (NMT) (Geng, Feng, and Qin 2021; Ghazvininejad et al. 2019; Huang, Perez, and Volkovs 2022; Saharia et al. 2020). These iterative decoding methods using different masking strategies, including heuristics based and and learning-based. Our locator is inspired from the latter approach. For a more complete review, please refer to the survey (Xiao et al. 2022).

Preliminary Study

Task Formulation

Layout generation can be viewed as a sequence generation:

$$\mathbf{s} = ([bos], c_1, x_1, y_1, w_1, h_1, ..., c_n, x_n, y_n, w_n, h_n, [eos])$$

where c_i is the category label of the i-th element in the layout (e.g., `title`, `text`, `figure`), x_i, y_i, w_i, h_i represent the position and size which are converted to discrete tokens. $[bos], [eos]$ are special tokens for beginning and end. The total sequence length is $5n + 2$.

Autoregressive (AR) generation predicts the token one at a time, conditioning on the previous generated sequence:

$$p(\mathbf{s}) = \prod_{i=2}^{5n+2} p(s_i | s_{1:i-1}) \tag{1}$$

while in the non-autoregressive setting, the model predicts one token with bidirectional attention and predicts the whole sequence simultaneously:

$$p(\mathbf{s}) = \prod_{i=2}^{5n+2} p(s_i | s_{1:i-1}, s_{i+1:5n+2}) \tag{2}$$

Non-Autoregressive Decoding Analysis

To better understand how different decoding methods (i.e., AR and NAR) behave in the layout synthesis process, this paper conducts an in-depth analysis with some interesting findings. In the following analysis, we use the state-of-the-art AR model LayoutTransformer as well as the the NAR model BLT on the scientific article dataset PubLaynet.

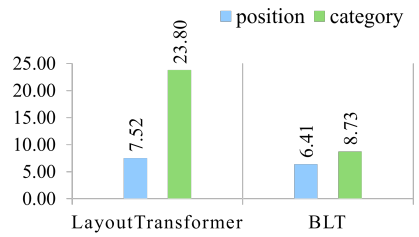

Figure 3: Comparison of AR (LayoutTransformer) and NAR (BLT) approaches using different input element orders (i.e., position, category). Smaller overlap degree indicates better performance.

Finding 1: NAR is robust to input element order. For sequence generation, the input order is an important factor. We compare the AR and NAR models with different element orders used in previous works (Yang et al. 2021; Kong et al. 2022): (1) position, where elements are sorted using the top-left coordinates. While most previous works follow this setting, it actually causes information leak of the ground truth data during inference since they use absolution positions in real layouts to determine the order; (2) category, where input elements are fed per category (e.g., generate all the `paragraph` elements first, then followed by `table`). Figure 3 shows the performances in terms of the `Overlap` metric, which calculates the average overlap degree between elements in a layout and is widely used for evaluation:

$$\text{Overlap} = \frac{1}{D} \sum_{d=1}^{D} \Big[\sum_{i=1}^{N} \sum_{j \neq i} \frac{e_i \cap e_j}{e_i} \Big] \tag{3}$$

where D denotes the number of layouts, N is the element number of a layout, e_i indicates the i-th element in a layout, $e_i \cap e_j$ means the overlap area between e_i and e_j.

Based on the general assumption that elements in a layout should not overlap excessively, smaller value indicates better quality. From the figure, we can see that the AR model is more sensitive to orders as its performance drops significantly from the 'category' setting to the 'position' one. Being compared, the NAR model is relatively robust to different input orders.

Finding 2: NAR exposes the issue of repetitive generation. Compared to AR with causal masked attention over previously generated tokens, NAR applies a full attention over all tokens on a wider range of contexts, which is difficult for the model to distinguish different tokens by solely relying on positional encodings. Thus it is more likely to generate repetitive tokens (Huang, Perez, and Volkovs 2022). Here we try to quantify this intrinsic error using the aforementioned `Overlap` metric. Higher overlap degree indicates that the model is more likely to generate tokens in a similar region. Table 1 shows the statistics considering overlaps between (1) elements within the same category and (2) all elements. We can see that the overlap problem in NAR is more serious than AR (26.89 compared to 23.80 for all elements), and nearly two times the degree under the 'same category' setting (15.84 compared to 8.67). Furthermore, we

Models	Iter	Overlap	
		same cat.	All
LayoutTransformer (AR)	/	8.67	23.80
BLT (NAR)	1	15.84	26.89
BLT	5	5.20	13.43
BLT	10	3.23	10.23

Table 1: NAR generates a larger percentage of repetitive tokens than the AR approach. The issue is alleviated with iterations of refinement.

Repre.	Noise range	Precision	Recall	F1 Score
Object	0.1	84.80	79.70	82.17
Pixel	0.1	92.19	88.78	90.45
Object	0.2	96.36	93.62	94.97
Pixel	0.2	96.56	95.73	96.14
Object	0.5	99.16	93.08	96.02
Pixel	0.5	97.35	95.58	96.46

Table 2: Classification results of our toy experiment to compare different layout representations (i.e., object and pixel space) using random noise data.

show that iterative refinement is an effective mechanism to alleviate the issue as the overlap (all) decreases from 26.89 (iter 1) to 10.23 (iter 10).

In this paper, we focus on the iteration-based NAR generation approach and propose a simple but effective locator module to improve the iterative refinement process.

Layout Representation: Object v.s. Pixel

Previous layout generation methods are far more based on primitive elements with attributes to form a sequence. Meanwhile, layouts can also be represented by a wireframe image which is rendered by the element attributes, which provides a more direct visual perception (Li et al. 2019). We argue that both representations are informative for capturing important features in different aspects.

To verify our claim, we design a toy experiment of detecting the erroneous element attributes in a noisy layout using the two different representations respectively. For training and evaluation, we sample elements in real layouts as good layouts, and add random noise on some attributes x, y, w, h as the bad layouts. In the object space given a sequence of element attributes, we adopt a Transformer with a classification layer on the top to predict good/bad for each token. In the pixel space we use the convolutional-based Faster-RCNN (Ren et al. 2015) to detect the inferior elements and classify each of its attributes. Table 2 shows the classification results. As we can see, when the noise range is large (e.g., 0.5), models in both spaces can detect accurately with F1 score over 96%. If the noise is decreased to a smaller range (e.g., 0.1) where the noisy layouts are less visually different from the real ones, models in pixel space performs significantly better. This indicates that pixel representation is more sensitive in capturing fine-grained spatial patterns such as minor misalignment and occlusion.

Our Approach

We generate graphic layouts using the non-autoregressive approach. Our model consists of a *decoder* to generate the tokens and a *locator* to recognize the erroneous tokens which will be revised by the decoder iteratively. Figure 2 shows the pipeline overview.

Decoder

Following BLT (Kong et al. 2022), we use a multi-layer transformer decoder to predict the layout sequence in parallel. Each token in the sequence is represented by the sum of its attribute embedding (i.e., category, position or size) and three additional position encodings γ to better distinguish different tokens, (1) the token index in a sentence; (2) the element-level index where tokens belonging to the same element will have the same value. (3) the number of elements where all tokens share the same value.

$$PE(i) = \gamma_1(i) + \gamma_2(\lfloor i/5 \rfloor) + \gamma_3(n) \tag{4}$$

where $i \in 5n + 2$.

During training, we randomly mask some tokens in the layout sequence $\mathbf{s}$ similar to BERT (Devlin et al. 2018) and compute the loss on these mask positions M to minimize the negative log-likelihood:

$$\mathcal{L}_{\mathbf{mask}} = -E_{\mathbf{s}\in\mathcal{D}}\Big[\sum_{i\in M} \log p(s_i|s^M)\Big] \tag{5}$$

In inference, we replace all the attributes which are not given with `[MASK]` token (in conditional generation, the element categories are given) to initialize the input sequence and the decoder is then used to predict the masked tokens.

Wireframe Conditioning. Given the wireframe image I encoded with a convoluational network (LeCun et al. 1998) φ, we apply the cross-attention of the Transformer-based decoder to the image, which provide the model a spatial context for learning to attend to relevant regions:

$$\text{Attention}(\mathbf{s}, I) = \text{softmax}\left(\frac{\phi(\mathbf{s})\varphi(I)^T}{\sqrt{d_k}}\right) \tag{6}$$

where $\phi(\mathbf{s})$ is the query matrix derived from the transformer decoder, $\varphi(I)$ is the key matrix from the convolutional network, respectively, and d_k is the dimension of the key $\varphi(I)$.

Locator

In the iteration-based NAR models, the dominant approach to identify the erroneous tokens for masking and re-predict is to employ heuristic rules to roughly choose the least confident tokens from decoder output. However, relying on the decoder's confidence score is likely to cause bias to its learned distribution, and the heuristic rules might not be sufficient to cover a wide range of cases.

This paper proposes to utilize a learning-based locator module to select the tokens. The locator consists of three

Algorithm 1: Training data construction for locator.

Require: Generated layouts L^g with n elements, ground truth layout L^{real} with the same categories $(c_0, c_1, \ldots, c_n)$, threshold δ

```
erroneous attribute set to be masked M[k] = {} for k in attributes {x, y, w, h}
Set matching {e_i^g} and {e_j^real} with distance function d
for i = 1, ··· , n do
    if abs(e_i^g[k] − e_j^real[k]) > δ then
        M[k] = M[k] + e_i^g[k]
    end if
end for
```

components: a backbone network, a mask classifier, and a bounding box predictor. Given the wireframe input I, the backbone network extracts features F_{feat} from I, the mask classifier binary classifies each of the attributes (x, y, w, h) to determine if it should be masked or not:

$$F_{feat} = \mathrm{Conv}(I), \tag{7}$$

$$C_{mask} = \mathbf{W_c}\mathrm{RPN}(F_{feat}) + \mathbf{b_c} \tag{8}$$

where $\mathbf{W_c}$ and $\mathbf{b_c}$ are the parameters of mask classifier.

To train the locator, there is no direct annotated data of noisy layout with targeted erroneous tokens. Here we introduce a novel algorithm to construct the training data, which is described in Algorithm 1. Specifically, we collect the generated noisy layouts from the trained decoder in different iterations. For each layout, we retrieve the ground truth layouts with the same element categories and conduct a bipartie matching between elements in the generated layout $\{e_i^g\}$ and the real one $\{e_j^r\}$. The matching function d considers the category constraint (element categories must be the same) and the spatial relationships (i.e., the bounding box overlap ratio, position distance, and the size disparities):

$$\begin{aligned} d(e_i^g, e_j^r) = \alpha_1 1_{c_i \neq c_j} + \alpha_2 \mathrm{IoU}(e_i^g, e_j^r) \\ + \alpha_3 f_{pos}(e_i^g, e_j^r) + \alpha_4 f_{area}(e_i^g, e_j^r), \end{aligned} \tag{9}$$

where f_{pos} and f_{area} are the functions which compute the position and area differences.

After matching, we choose elements whose overlap with the ground truth smaller than a threshold as the mistakes and annotate its corresponding attributes to be masked.

Experiments

Dataset

We experiment with two widely used public datasets in different design types: Rico (Deka et al. 2017; Liu et al. 2018) and Publaynet (Zhong, Tang, and Yepes 2019). **Rico** is a UI/UX design dataset consisting of over 66k UI layouts collected from Android mobile apps. We filter out layouts with the number of elements in a layout more than 9 and use the most common 13 category in Rico following the previous work (Kikuchi et al. 2021). There are 20,606 layouts in total finally. We split train/val/test dataset at a rate of 0.85/0.05/0.10. **PubLaynet** contains over 360k layouts of scientific paper pages from PubMed Central. We use the official split train and test set. Similarly, we exclude layouts elements more than 9, which are in total 173,225 layouts. For inference, we use the 'category' input order mentioned in previous section for conditional generation.

Evaluation Metrics

There are 4 metrics commonly used to measure the generated layout quality:

- **Maximum IoU.** Given the generated layouts and the references, this metric computes the intersection over the union of the two sets with a permutation to maximize the IoU as a similarity measurement.
- **Alignment.** Layout elements are usually aligned with each other to create an organized composition. Alignment calculates on average the minimum distance in the x- or y-axis between any element pairs in a layout.
- **Overlap.** It is assumed that elements should not overlap excessively. Overlap computes the average IoU of any two elements in a layout. Layouts with small overlap values are often considered to be high quality.
- **FID.** Compared to the above heuristic metrics, FID is a sample-based metric for image generation (Heusel et al. 2017) and has been adopted in layout generation. It pretrains a feature network to classify real or fake layouts which is then used to extract features of two data sets and calculate the Fréchet distance. Here we use two pretrained FID, namely SeqFID (Kikuchi et al. 2021) and PixelFID (Yang et al. 2023) using different layout representations (object and pixel respectively). Please refer to Yang et al. (2023) for a more comprehensive comparison between the two metrics.

Baselines

We consider the following public-available works as baselines, including the **autoregressive** models: **LayoutVAE** (Jyothi et al. 2019) takes the latent code and category labels (optional) as input and generates the element bounding boxes in an autoregressive manner. **VTN** (Arroyo, Postels, and Tombari 2021) uses transformer layer to enhance the performance of VAE as well as increase the number of elements in layouts. **LayoutGAN++** (Kikuchi et al. 2021) improves LayoutGAN with Transformer backbone and applies several beautification post-process for alignment and non-overlap. **LayoutTransformer** (Yang et al. 2021) autoregressively generates a sequence of element tokens. And we also compare with the state-of-the-art **non-autoregressive** model BLT (Kong et al. 2022). Specifically, they apply a heuristic strategy **HSP** for iterative decoding: attributes are grouped into category, size and position, and tokens in different groups are predicted in a pre-defined order. In our experiment, we observe that such strategy is not always effective so we also show the result **BLT w/o HSP**.

Main Results

Following BLT, we consider two settings of conditional generation: (1) Condition on Category (C→SP), which only element categories are given as input and the model needs to

Models	PubLaynet					Rico				
	SeqFID↓	PixelFID↓	Max IoU↑	Overlap↓	Alignment↓	SeqFID↓	PixelFID↓	Max IoU↑	Overlap↓	Alignment↓
Layout VAE	27.18	158.72	0.2473	45.81	0.6617	87.61	12.52	0.3678	63.20	0.8709
LayoutTransformer	10.20	76.96	0.3465	23.80	0.0912	44.77	16.31	0.4165	**61.04**	**0.0731**
VTN	8.60	78.76	0.3122	26.51	0.2196	18.49	28.34	0.4332	63.22	0.4411
Layout GAN++	9.52	53.71	**0.4827**	14.50	0.2019	**13.49**	6.78	0.5038	65.14	0.3087
BLT	7.88	4.60	0.3865	8.73	0.0765	32.67	3.80	0.5669	72.64	0.2361
BLT w/o HSP	**6.00**	3.82	0.3985	9.21	0.0988	32.06	3.36	0.5773	68.66	0.2152
Ours (dec-only)	6.30	2.52	0.3959	8.64	0.0740	31.21	**2.65**	**0.5807**	66.33	0.1995
Ours (w/ locator)	9.99	**2.33**	0.2962	**7.57**	**0.0634**	27.04	2.82	0.5693	62.25	0.2075
Real data	4.91	0.02	0.5300	0.22	0.0400	4.26	1.16	0.6600	48.43	0.2000

Table 3: Conditional generation results when given element category (C→SP). The best results are in bold.

Models	PubLaynet					Rico				
	SeqFID↓	PixelFID↓	Max IoU↑	Overlap↓	Alignment↓	SeqFID↓	PixelFID↓	Max IoU↑	Overlap↓	Alignment↓
LayoutVAE	22.93	134.90	0.3295	45.59	0.6257	86.64	7.71	0.4124	63.79	0.4899
LayoutTransformer	26.49	130.77	0.3131	36.04	0.3261	59.81	11.77	0.4262	**59.42**	**0.1737**
BLT	2.39	3.04	**0.5502**	8.96	0.1404	20.07	2.26	0.5737	69.31	0.3941
BLT w/o HSP	2.27	1.60	0.5493	8.47	0.1356	18.08	2.17	0.5800	64.73	0.2858
Ours (dec-only)	2.09	1.51	0.5450	**8.44**	**0.1275**	19.42	1.91	0.5774	65.24	0.2647
Ours (w/ locator)	**2.04**	**1.44**	0.5428	9.69	0.1280	**17.88**	**1.88**	**0.5801**	62.47	0.3070
Real data	4.91	0.02	0.5300	0.22	0.0400	4.26	1.16	0.6600	48.43	0.2000

Table 4: Conditional generation results given element category and size (CS→P). The best results are in bold.

predict the element size and position; (2) Condition on Category + Size (CS→P), which the element category and size are specified and the model predicts only the positions. We show the performances in Table 3 and Table 4 respectively[1].

As we can see, the NAR baseline BLT can already achieve similar performances as the AR approaches, and even significantly better on PubLaynet under both C→SP and C→SP settings. For our approach, only using the wireframe-conditioned decoder (dec-only) achieves promising results (e.g., PixelFID 2.52 compared to 4.60 in BLT on PubLaynet under C→SP setting). Coupled with our learning-based locator, we reach the best results in most of the metrics. Please note that for the two FID evaluators, we see similar trends in relative comparison between different systems, but PixelFID is proved to be more robust and unbiased to different decoder architecture (Yang et al. 2023).

Qualitative Results

Here we show some generated layouts from the baselines (i.e., LayoutTransformer, BLT) and our approach, as well as the real layouts, in Figure 4. We can observe a typical error in the AR model LayoutTransformer that when it generates an inferior element of large area, it is impossible to revise and thus the subsequent elements can only be placed over, resulting in bad occlusion. Similar, BLT also suffers from the occlusion issue which might caused by the simple but weak heuristics that detect the erroneous tokens. Being compared, our model generates more balanced layouts that elements are evenly distributed, which indicates that our model has a stronger capability of capturing global contexts and locating mistakes. Moreover, our generated results enjoys better alignment and less overlap in most cases.

[1]Due to space limit, we show the unconditional task in Suppl.

Model Analysis

To better understand what our model has learned, this subsection conducts several ablation studies on different model components. The following experiments are conducted on PubLaynet. In addition, we show more results in the suppl.

Performance along Iterations. Figure 6 shows the overlap performance of BLT and our model in different iterations. As we can see, our model gets significantly improved from iteration 1 to 10 (smaller overlap is better), which means that the iterative decoding with our learning-based locator is effective to recognize erroneous element attributes and refine the layouts. Furthermore, our model gets faster convergence than BLT, achieving stable results in around iteration 6. As the number of iterations increases, the performance gain is becoming smaller, which is expected since the generated layouts have been refined to be better after several iterations of mask-predict.

Locator Ablations. This experiment investigates the impacts of different locator settings in Table 5. Besides the end-to-end results, we show the intermediate detection accuracy of the locator. Here we obtain two interesting observations:

(1) pixel space is more informative than object space for capturing spatial patterns. Similar to the previous toy experiment in Table 2, we try the two data representation for locator input. To exclude the factor of model back-

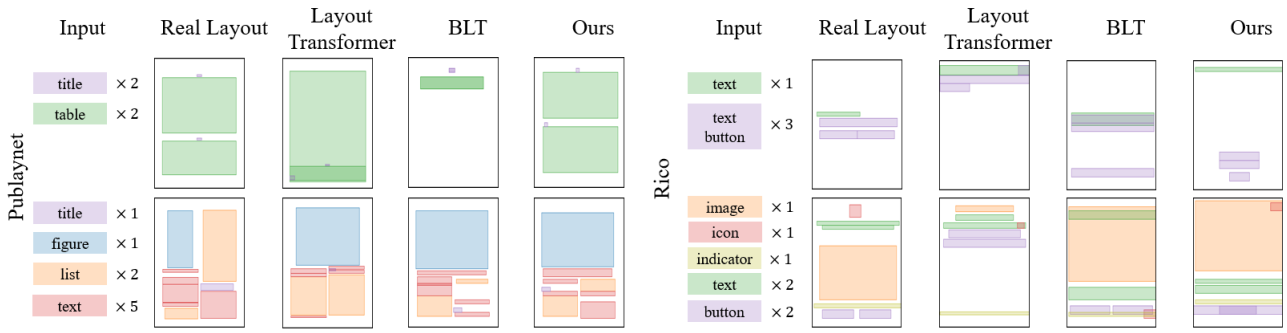

Figure 4: Qualitative results on Publaynet and Rico.

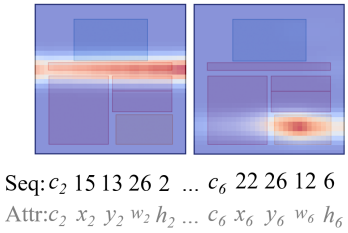

Figure 5: Decoder-Wireframe cross-attn visualization.

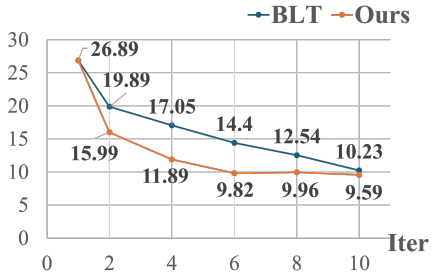

Figure 6: Overlap performance of BLT and our models with different decoding iterations.

bone and ensure a fairer comparison, we try different locator backbones (i.e., Convolution-based Faster-RCNN and Transformer-based DETR (Carion et al. 2020)) with similar parameter size. As we can see that different backbones in the pixel space perform comparably well, and are both significantly better than the one in object space. Specifically, locator in pixel space (FasterRCNN) detects the inferior elements more accurately (f1 score 59.82 compared to 29.52 in object space setting), and thus results in a better end-to-end results. This further indicates that pixel space can capture more informative spatial patterns in the layout.

(2) training data distribution matters for aligning the locator and the decoder. As previously mentioned, we train the locator using the decoder generated outputs, which aims to align better with the decoder's distribution. To support our argument, we use a different training set for locator where we add random noise to the element attributes in real layouts, which is the same as the data construction in Table 2.

Settings	cls. p	cls. r	cls. f1	e2e PixelFID↓	e2e Overlap↓
Dec-only	-	-	-	113.36	26.89
Object	35.55	25.24	29.52	14.11	11.11
Pixel (FasterRCNN)	**62.75**	57.14	**59.82**	**2.33**	**7.57**
Pixel (DETR)	36.10	**98.53**	52.84	3.12	7.71
Dec-output	**62.75**	**57.14**	**59.82**	**2.33**	**7.57**
RandNoise	20.40	53.45	29.53	19.23	19.69

Table 5: Performances under different locator settings, including input layout representation (Object, Pixel), backbone architecture (FasterRCNN, DETR), training data source (Dec-Output, RandNoise).

From the last two rows in Table 5, we can see that using decoder-output data (Dec-output) is more effective than the one using random-noise data (RandNoise), indicating that the distribution alignment can help locator better focus on the decoder's error cases and result in higher performance.

Visualization of Decoder-Wireframe Cross-Attention. Different from BLT, our decoder is conditioned on a wireframe image which is rendered from the decoder's output sequence in the previous iteration. To investigate what the model has captured, we visualize the cross attention weight (Equation 6) in Figure 5.Both the two elements' category tokens have correctly attended the corresponding regions in the wireframe, which means that the decoder can effectively utilize contexts in pixel space for better generation.

Conclusion

NAR layout generation shows competitive results compared to the AR approach. In this paper, we conduct a detailed analysis to compare the two frameworks, as well as different layout representations (i.e., object and pixel). Based on our observation, we propose a wireframe locator which explicitly learns to detect the erroneous tokens with visual input. Experiments show that our approach can achieve remarkable improvements over previous models. Further analysis reveals patterns that our model has learned. In the future, we would like to extend to content-aware generation. How to distill the knowledge in large-scale multimodal model for graphic design is another interesting direction to explore.

References

Arroyo, D. M.; Postels, J.; and Tombari, F. 2021. Variational Transformer Networks for Layout Generation. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 13637–13647. IEEE.

Cao, Y.; Ma, Y.; Zhou, M.; Liu, C.; Xie, H.; Ge, T.; and Jiang, Y. 2022. Geometry Aligned Variational Transformer for Image-conditioned Layout Generation. In *Proceedings of the 30th ACM International Conference on Multimedia*, 1561–1571. Lisboa Portugal: ACM. ISBN 978-1-4503-9203-7.

Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A.; and Zagoruyko, S. 2020. End-to-end object detection with transformers. In *European conference on computer vision*, 213–229. Springer.

Deka, B.; Huang, Z.; Franzen, C.; Hibschman, J.; Afergan, D.; Li, Y.; Nichols, J.; and Kumar, R. 2017. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th annual ACM symposium on user interface software and technology*, 845–854.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Feng, G.; Huang, D.; Lin, C.-Y.; Dakic, D.; Milunovic, M.; Stankovic, T.; and Ilic, I. 2022. Exploring Heterogeneous Feature Representation for Document Layout Understanding. In *Proceedings of the 34th IEEE International Conference on Tools with Artificial Intelligence*.

Geng, X.; Feng, X.; and Qin, B. 2021. Learning to Rewrite for Non-Autoregressive Neural Machine Translation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 3297–3308. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics.

Ghazvininejad, M.; Levy, O.; Liu, Y.; and Zettlemoyer, L. 2019. Mask-Predict: Parallel Decoding of Conditional Masked Language Models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 6112–6121. Association for Computational Linguistics.

Guo, M.; Huang, D.; and Xie, X. 2021. The Layout Generation Algorithm of Graphic Design Based on Transformer-CVAE. *2021 International Conference on Signal Processing and Machine Learning (CONF-SPML)*, 219–224.

Hao, X.; Huang, D.; Lin, J.; and Lin, C.-Y. 2023. Relation-enhanced DETR for Component Detection in Graphic Design Reverse Engineering. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, 4785–4793.

Heusel, M.; Ramsauer, H.; Unterthiner, T.; Nessler, B.; and Hochreiter, S. 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30.

Huang, D.; Guo, J.; Sun, S.; Tian, H.; Lin, J.; Hu, Z.; Lin, C.-Y.; Lou, J.-G.; and Zhang, D. 2023. A Survey for Graphic Design Intelligence. *arXiv preprint arXiv:2309.01371*.

Huang, D.; Liu, J.; Lin, C.-Y.; and Yin, J. 2018a. Neural math word problem solver with reinforcement learning. In *Proceedings of the 27th International Conference on Computational Linguistics*, 213–223.

Huang, D.; Wang, J.; Wang, G.; and Lin, C.-Y. 2021. Visual style extraction from chart images for chart restyling. In *2020 25th International Conference on Pattern Recognition (ICPR)*, 7625–7632. IEEE.

Huang, D.; Yao, J.-G.; Lin, C.-Y.; Zhou, Q.; and Yin, J. 2018b. Using Intermediate Representations to Solve Math Word Problems. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 419–428. Association for Computational Linguistics.

Huang, X. S.; Perez, F.; and Volkovs, M. 2022. Improving Non-Autoregressive Translation Models Without Distillation. In *Proceedings of the Eleventh International Conference on Learning Representations*.

Hui, M.; Zhang, Z.; Zhang, X.; Xie, W.; Wang, Y.; and Lu, Y. 2023. Unifying Layout Generation with a Decoupled Diffusion Mode. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 1942–1951.

Hurst, N.; Li, W.; and Marriott, K. 2009. Review of automatic document formatting. In *Proceedings of the 9th ACM symposium on Document engineering*, 99–108.

Inoue, N.; Kikuchi, K.; Simo-Serra, E.; Otani, M.; and Yamaguchi, K. 2023. LayoutDM: Discrete Diffusion Model for Controllable Layout Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 10167–10176.

Jiang, Z.; Sun, S.; Zhu, J.; Lou, J.-G.; and Zhang, D. 2022. Coarse-to-Fine Generative Modeling for Graphic Layouts. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(1): 1096–1103.

Jyothi, A. A.; Durand, T.; He, J.; Sigal, L.; and Mori, G. 2019. LayoutVAE: Stochastic Scene Layout Generation From a Label Set. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 9894–9903.

Kikuchi, K.; Simo-Serra, E.; Otani, M.; and Yamaguchi, K. 2021. Constrained Graphic Layout Generation via Latent Optimization. In *Proceedings of the 29th ACM International Conference on Multimedia*, 88–96. ArXiv: 2108.00871.

Kong, X.; Jiang, L.; Chang, H.; Zhang, H.; Hao, Y.; Gong, H.; and Essa, I. 2022. BLT: Bidirectional Layout Transformer Fornbsp;Controllable Layout Generation. In *Computer Vision – ECCV 2022: 17th European Conference*, 474–490.

Kumar, R.; Talton, J. O.; Ahmad, S.; and Klemmer, S. R. 2011. Bricolage: example-based retargeting for web design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2197–2206.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324.

Lee, H.-Y.; Jiang, L.; Essa, I.; Le, P. B.; Gong, H.; Yang, M.-H.; and Yang, W. 2020. Neural Design Network: Graphic

Layout Generation with Constraints. In *Computer Vision – ECCV 2020: 16th European Conference, 2020, Proceedings, Part III*, 491–506. Springer-Verlag.

Li, C.; Zhang, P.; and Wang, C. 2022. Harmonious Textual Layout Generation Over Natural Images via Deep Aesthetics Learning. *IEEE Transactions on Multimedia*, 24: 3416–3428. Conference Name: IEEE Transactions on Multimedia.

Li, D.-W.; Huang, D.; Ma, T.; and Lin, C.-Y. 2021. Towards topic-aware slide generation for academic papers with unsupervised mutual learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 13243–13251.

Li, J.; Yang, J.; Hertzmann, A.; Zhang, J.; and Xu, T. 2019. LayoutGAN: Generating Graphic Layouts with Wireframe Discriminators. In *ICLR*. ArXiv: 1901.06767.

Liu, T. F.; Craft, M.; Situ, J.; Yumer, E.; Mech, R.; and Kumar, R. 2018. Learning design semantics for mobile apps. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, 569–579.

O'Donovan, P.; Agarwala, A.; and Hertzmann, A. 2014. Learning layouts for single-pagegraphic designs. *IEEE transactions on visualization and computer graphics*, 20(8): 1200–1213.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, 91–99. MIT Press.

Saharia, C.; Chan, W.; Saxena, S.; and Norouzi, M. 2020. Non-Autoregressive Machine Translation with Latent Alignments. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1098–1108. Online: Association for Computational Linguistics.

Shi, D.; Cui, W.; Huang, D.; Zhang, H.; and Cao, N. 2023. Reverse-engineering information presentations: recovering hierarchical grouping from layouts of visual elements. *Visual Intelligence*, 1(1): 1–14.

Tabata, S.; Yoshihara, H.; Maeda, H.; and Yokoyama, K. 2019. Automatic Layout Generation for Graphical Design Magazines. SIGGRAPH '19. Association for Computing Machinery.

Vaddamanu, P.; Aggarwal, V.; Guda, B. P. R.; Srinivasan, B. V.; and Chhaya, N. 2022. Harmonized Banner Creation from Multimodal Design Assets. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, 1–7. New Orleans LA USA: ACM. ISBN 978-1-4503-9156-6.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Wang, Y.; Pu, G.; Luo, W.; Wang, Y.; Xiong, P.; Kang, H.; and Lian, Z. 2022. Aesthetic Text Logo Synthesis via Content-aware Layout Inferring. *arXiv:2204.02701 [cs]*. ArXiv: 2204.02701.

Weng, H.; Huang, D.; Zhang, T.; and Lin, C.-Y. 2023. Learn and Sample Together: Collaborative Generation for Graphic Design Layout. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, 5851–5859.

Xiao, Y.; Wu, L.; Guo, J.; Li, J.; Zhang, M.; Qin, T.; and Liu, T. 2022. A Survey on Non-Autoregressive Generation for Neural Machine Translation and Beyond. *CoRR*, abs/2204.09269.

Xie, Y.; Huang, D.; Wang, J.; and Lin, C.-Y. 2021. CanvasEmb: Learning Layout Representation with Large-Scale Pre-Training for Graphic Design. In *Proceedings of the 29th ACM international conference on multimedia*, 4100–4108.

Yang, C.-F.; Fan, W.-C.; Yang, F.-E.; and Wang, Y.-C. F. 2021. LayoutTransformer: Scene Layout Generation with Conceptual and Spatial Diversity. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 3731–3740. IEEE.

Yang, H.; Huang, D.; Lin, C.-Y.; and He, S. 2023. Layout Generation as Intermediate Sequence Generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zhang, J.; Guo, J.; Sun, S.; Lou, J.-G.; and Zhang, D. 2023. LayoutDiffusion: Improving Graphic Layout Generation by Discrete Diffusion Probabilistic Models. In *ICCV 2023*.

Zheng, X.; Qiao, X.; Cao, Y.; and Lau, R. W. H. 2019. Content-aware generative modeling of graphic design layouts. *ACM Transactions on Graphics*, 38(4): 1–15.

Zhong, X.; Tang, J.; and Yepes, A. J. 2019. Publaynet: largest dataset ever for document layout analysis. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, 1015–1022. IEEE.

Zhou, M.; Xu, C.; Ma, Y.; Ge, T.; Jiang, Y.; and Xu, W. 2022. Composition-aware Graphic Layout GAN for Visual-Textual Presentation Designs. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, 4995–5001. Vienna, Austria: International Joint Conferences on Artificial Intelligence Organization. ISBN 978-1-956792-00-3.

Zhou, Q.; and Huang, D. 2019. Towards generating math word problems from equations and topics. In *Proceedings of the 12th International Conference on Natural Language Generation*, 494–503.

Zhu, J.; Huang, D.; Wang, C.; Cheng, M.; Li, J.; Hu, H.; Geng, X.; and Guo, B. 2024. Unsupervised Graphic Layout Grouping with Transformers. In *Proceedings of IEEE/CVF Winter Conference on Applications of Computer Vision*.